**Pintos Project 0-2: Pintos Data Structure**

|  |  |
| --- | --- |
| 담당 교수 : | 김영재 |
| 학번 : | 20161640 |
| 이름 : | 정동혁 |
|  |  |

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | bool list\_less\_func\_new(const struct list\_elem \*, const struct list\_elem \*, void \*aux) |
| **Parameter** | 1. 비교할 첫번째 list\_elem 2. 비교할 두번째 list\_elem 3. 보조 parameter |
| **Return** | bool : true(두번째 list\_elem이 클 때), false(두번째 list\_elem이 작을 때) |
| **Function** | list정렬을 위한 대소비교를 하는 함수로서 파라미터인 list\_elem에 list\_entry를 이용해 각각의 list\_item을 받아와 data값을 비교해준다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_hash\_func\_new(const struct hash\_elem \*, void \*aux) |
| **Parameter** | 1. 해시값을 가져올 hash\_elem 2. 보조 parameter |
| **Return** | unsigned (hash value) |
| **Function** | 입력받은 hash\_elem에 hash\_entry를 통해서 data를 가져온뒤에 해시함수의 parameter로 넣어주어 해시함수의 결과를 return해준다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_less\_func\_new(const struct hash\_elem \*, const struct hash\_elem \*, void \*aux) |
| **Parameter** | 1. 비교할 첫번째 hash\_elem 2. 비교할 두번째 hash\_elem 3. 보조 parameter |
| **Return** | bool : true(두번째 hash\_elem이 클 때), false(두번째 hash\_elem이 작을 때) |
| **Function** | hash정렬을 위한 대소비교를 하는 함수로서 파라미터인 hash\_elem에 hash\_entry를 이용해 각각의 hash\_item을 받아와 data값을 비교해준다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_action\_func\_des(struct hash\_elem \*, void \*aux) |
| **Parameter** | 1. 제거해줄 hash\_elem 2. 보조 parameter |
| **Return** | void |
| **Function** | 입력받은 hash\_elem을 해쉬테이블 내부에서 제거해준다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_action\_func\_square(struct hash\_elem \*e, void \*aux) |
| **Parameter** | 1. 값을 제곱해줄 hash\_elem 2. 보조 parameter |
| **Return** | void |
| **Function** | 입력받은 hash\_elem에 hash\_entry를 통해 hash\_item을 받아와준뒤에 data값을 제곱해준다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_action\_func\_triple(struct hash\_elem \*e, void \*aux) |
| **Parameter** | 1. 값을 세제곱해줄 hash\_elem 2. 보조 parameter |
| **Return** | void |
| **Function** | 입력받은 hash\_elem에 hash\_entry를 통해 hash\_item을 받아와준뒤에 data값을 세제곱해준다. |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | #define list\_entry(LIST\_ELEM, STRUCT, MEMBER) \  ((STRUCT \*) ((uint8\_t \*) &(LIST\_ELEM)->next \  - offsetof (STRUCT, MEMBER.next))) |
| **Parameter** | 1. list\_item을 얻고싶은 list\_elem 2. struct list\_item(내장되어있는 구조) 3. elem(member) |
| **Return** | 입력받은 list\_elem이 내장되어있는 list\_item struct(2nd parameter) |
| **Function** | 1st parameter로 list\_elem을 입력받고 outer structure인 2nd parameter와 member인 3rd parameter를 통해 그 list\_elem이 내장되어 있는 2nd parameter형태를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool  is\_head (struct list\_elem \*elem) |
| **Parameter** | 1. head인지 확인할 list\_elem |
| **Return** | static inline bool : true(1st parameter가 head인 경우), false(1st parameter가 head가 아닌 경우) |
| **Function** | list\_elem을 입력으로 받아 list의 head면 true를 반환하고, 아닐 경우 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool  is\_interior (struct list\_elem \*elem) |
| **Parameter** | 1. list의 내부에 존재하는 list\_elem인지 확인할 list\_elem |
| **Return** | static inline bool : true(1st parameter가 list내부 list\_elem인 경우), false(1st parameter가 list내부 list\_elem이 아닌 경우) |
| **Function** | list\_elem을 입력으로 받아 list내부의 list\_elem이면 true를 반환하고, 아닐 경우 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static inline bool  is\_tail (struct list\_elem \*elem) |
| **Parameter** | 1. tail인지 확인할 list\_elem |
| **Return** | static inline bool : true(1st parameter가 tail인 경우), false(1st parameter가 tail이 아닌 경우) |
| **Function** | list\_elem을 입력으로 받아 list의 tail이면 true를 반환하고, 아닐 경우 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_init (struct list \*); |
| **Parameter** | 1. initial작업을 해줄 list |
| **Return** | void |
| **Function** | list를 초기화 해주는 함수로 head와 tail을 설정해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_begin (struct list \*) |
| **Parameter** | 1. head다음의 첫번째 위치를 찾아줄 list |
| **Return** | struct list\_elem \* (입력받은 list의 head다음 list\_elem) |
| **Function** | 입력받은 list의 head(맨앞)다음에 있는 list\_elem을 찾아 반환해준다. 여기서 구한 첫번째 list\_elem은 특정 list의 list\_elem iteration을 위한 초기조건으로 사용된다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_next (struct list\_elem \*) |
| **Parameter** | 1. 다음 list\_elem을 구하기 위한 현재 list\_elem |
| **Return** | struct list\_elem \* (다음 list\_elem) |
| **Function** | 입력받은 list\_elem을 통해 다음 list\_elem을 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_end (struct list \*) |
| **Parameter** | 1. 마지막 위치(tail)를 찾아줄 list |
| **Return** | struct list\_elem \* (입력받은 list의 맨마지막(tail)) |
| **Function** | 입력받은 list의 맨 마지막에 있는 list\_elem(tail)을 찾아 반환해준다. 여기서 구한 마지막 list\_elem은 특정 list의 list\_elem을 begin부터 iteration작업을 할 때 종료조건으로 사용된다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rbegin (struct list \*) |
| **Parameter** | 1. tail이전의 마지막 list\_elem을 찾아줄 list |
| **Return** | struct list\_elem \* (입력받은 list의 tail이전 마지막 list\_elem) |
| **Function** | 입력받은 list의 맨 마지막 tail 이전에 있는 list\_elem을 구해준다. 여기서 list\_end와의 차이점은 list\_end는 tail을 반환해주는 반면 list\_rbegin은 tail바로 이전의 마지막으로 데이터를 갖는 list\_elem을 반환해준다. 즉, list의 reverse order의 list\_begin의 역할을 해준다고 할 수 있다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_prev (struct list\_elem \*) |
| **Parameter** | 1. 이전 list\_elem을 구하기 위한 현재 list\_elem |
| **Return** | struct list\_elem \* (이전 list\_elem) |
| **Function** | 입력받은 list\_elem을 통해 이전 list\_elem을 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_rend (struct list \*) |
| **Parameter** | 1. 첫번째 위치(head)를 찾아줄 list |
| **Return** | struct list\_elem \* (입력받은 list의 head) |
| **Function** | 입력받은 list의 맨 처음에 있는 list\_elem(head)을 찾아 반환해준다. 여기서 list\_begin과의 차이점은 list\_begin은 head다음 list\_elem을 반환해주는 반면 list\_rend는 head를 반환해준다. 즉, list의 reverse order의 list\_end의 역할을 해준다고 할 수 있다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_head (struct list \*) |
| **Parameter** | 1. 첫번째 위치(head)를 찾아줄 list |
| **Return** | struct list\_elem \* (입력받은 list의 head) |
| **Function** | 입력받은 list의 head를 반환해주는 함수. list\_rend와 코드, 기능은 같다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_tail (struct list \*) |
| **Parameter** | 1. 마지막 위치(tail)를 찾아줄 list |
| **Return** | struct list\_elem \* (입력받은 list의 맨마지막(tail)) |
| **Function** | 입력받은 list의 tail을 반환해주는 함수. list\_end와 코드, 기능은 같다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert (struct list\_elem \*, struct list\_elem \*) |
| **Parameter** | 1. 이전공간에 insert를 해야하는 기준 list\_elem 2. 특정위치에 insert를 해줄 list\_elem |
| **Return** | void |
| **Function** | 특정 위치에 있는 list\_elem(1st parameter) 앞에 한 개의 list\_elem(2nd parameter)을 insert해주고 원래있던 list\_elem들을 한칸씩 밀어주는 역할을 한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_splice (struct list\_elem \*before,  struct list\_elem \*first, struct list\_elem \*last) |
| **Parameter** | 1. 다음공간에 insert를 해야하는 기준 list\_elem 2. insert를 해줄 list의 시작점 위치의 list\_elem 3. insert를 해줄 list의 마지막 위치의 list\_elem |
| **Return** | void |
| **Function** | 어떠한 list 내부의 특정 위치에 있는 list\_elem(1st parameter) 다음에 또다른 list 내부의 특정 위치의 시작점(2nd parameter)부터 마지막위치(3rd parameter)까지의 list\_elem들을 insert해준다. 이후에 insert해준 list\_elem들을 가지고 있던 list내부의 list\_elem들은 삭제된다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_front (struct list \*, struct list\_elem \*) |
| **Parameter** | 1. list\_elem을 front에 push해줄 list 2. push해줄 list\_elem |
| **Return** | void |
| **Function** | 2nd parameter인 list\_elem을 list\_insert(list\_begin(1st), 2nd) 함수를 이용하여 1st parameter인 list의 begin지점 앞에 insert해줌으로서 list의 head다음에 있는 첫번째 list\_elem으로 만들어준다. 즉, 1st parameter인 list의 list\_begin으로 만들어준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_push\_back (struct list \*, struct list\_elem \*) |
| **Parameter** | 1. list\_elem을 back에 push해줄 list 2. push해줄 list\_elem |
| **Return** | void |
| **Function** | 2nd parameter인 list\_elem을 list\_insert(list\_end(1st), 2nd) 함수를 이용하여 1st parameter인 list의 end(tail)지점 바로 앞에 insert해줌으로서 list의 tail이전의 마지막 list\_elem으로 만들어준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_remove (struct list\_elem \*) |
| **Parameter** | 1. list에서 삭제려는 특정 list\_elem |
| **Return** | struct list\_elem \* (삭제된 list\_elem의 다음 list\_elem) |
| **Function** | list내부에서 입력받은 list\_elem을 prev와 next pointer를 이용하여 list내부에 존재하지 않게 해주고 그 다음 list\_elem을 반환한다. 만약 list내부에 입력받은 list\_elem이 존재하지 않을 경우 작동하지 않는다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_front (struct list \*) |
| **Parameter** | 1. 첫번째 list\_elem을 제거할 list |
| **Return** | struct list\_elem \* (제거된 list\_elem) |
| **Function** | 입력받은 list내부에서 list\_front를 통해 head 바로 다음의 list\_elem에 접근한뒤에 제거해주고, 이를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_pop\_back (struct list \*) |
| **Parameter** | 1. 마지막 list\_elem을 제거할 list |
| **Return** | struct list\_elem \* (제거된 list\_elem) |
| **Function** | 입력받은 list내부에서 list\_back을 통해 tail 바로 이전의 list\_elem에 접근한뒤에 제거해주고, 이를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_front (struct list \*) |
| **Parameter** | 1. head 다음의 list\_elem을 찾아줄 list |
| **Return** | struct list\_elem \* (head 다음에 존재하는 list\_elem) |
| **Function** | 입력받은 list의 head 바로 다음의 list\_elem을 next포인터를 통해 찾고, 이를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_back (struct list \*) |
| **Parameter** | 1. tail 이전의 list\_elem을 찾아줄 list |
| **Return** | struct list\_elem \* (tail 이전에 존재하는 list\_elem) |
| **Function** | 입력받은 list의 tail 바로 이전의 list\_elem을 prev포인터를 통해 찾고, 이를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t list\_size (struct list \*) |
| **Parameter** | 1. size를 구할 list |
| **Return** | size\_t(unsigned int) (list내부의 list\_elem의 개수) |
| **Function** | 입력받은 list내부의 list\_elem개수를 iteration을 통해 구하고 이를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool list\_empty (struct list \*) |
| **Parameter** | 1. 비어있는지의 여부를 결정할 list |
| **Return** | bool : true(list내부에 list\_elem이 하나도 없을 때), false(list내부에 list\_elem이 하나라도 있을 때) |
| **Function** | 입력받은 list내부의 list\_begin과 list\_end함수를 통해 이들이 같다면 true를 반환, 다르다면 false를 반환한다. 즉, list가 비어있는지를 확인해준다. |

|  |  |
| --- | --- |
| **Prototype** | static void  swap (struct list\_elem \*\*a, struct list\_elem \*\*b) |
| **Parameter** | 1. 바꿔줄 list\_elem pointer 2. 바꿔줄 list\_elem pointer |
| **Return** | static void |
| **Function** | struct list\_elem \* type을 주소값을 이용하여 swap해준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_reverse (struct list \*) |
| **Parameter** | 1. 내부의 list\_elem들을 reverse할 list |
| **Return** | void |
| **Function** | 입력받은 list내부의 list\_elem들을 swap function과 iteration을 통해 reverse order로 바꿔준다. |

|  |  |
| --- | --- |
| **Prototype** | static bool  is\_sorted (struct list\_elem \*a, struct list\_elem \*b,  list\_less\_func \*less, void \*aux) |
| **Parameter** | 1. 비교할 list\_elem 2. 비교할 list\_elem 3. list\_elem간의 대소비교를 위한 function 4. 보조 parameter |
| **Return** | static bool : true(iteration을 통해 1st parameter가 2nd parameter에 도달하기 전까지 3rd parameter의 정렬기준을 만족하는 경우), false(3rd parameter function의 parameter로 1st parameter와 바로 이전의 list\_elem을 차례대로 넣었을 때 false값을 반환하는경우) |
| **Function** | iteration을 통해 1st parameter와 그 바로 이전의 list\_elem을 3rd parameter function에 차례대로 parameter로 넣었을 때 a가 2nd parameter에 도달하는 경우(정렬기준 만족) true를 반환하고, 하나의 list\_elem이라도 정렬기준을 만족하지 못하는경우 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static struct list\_elem \*  find\_end\_of\_run (struct list\_elem \*a, struct list\_elem \*b,  list\_less\_func \*less, void \*aux) |
| **Parameter** | 1. 검색시 list\_elem의 시작점 2. 검색시 list\_elem의 끝점  3. list\_elem간의 대소비교를 위한 function 4. 보조 parameter |
| **Return** | static struct list\_elem \* (less function에 의해서 1st parameter와 바로이전의 list\_elem과 비교했을 때 정렬기준을 만족하지 못하는 list\_elem) |
| **Function** | iteration을 통해 1st parameter와 그 바로 이전의 list\_elem을 3rd parameter function에 차례대로 parameter로 넣었을 때 만족하지 못하는 경우가 나오면 loop를 종료하고 만족하지 못하는 list\_elem을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static void  inplace\_merge (struct list\_elem \*a0, struct list\_elem \*a1b0,  struct list\_elem \*b1,  list\_less\_func \*less, void \*aux) |
| **Parameter** | 1. 첫번째 병합 list\_elem 2. 중간 병합 list\_elem 3. 마지막 병합 list\_elem  4. list\_elem간의 대소비교를 위한 function 5. 보조 parameter |
| **Return** | static void |
| **Function** | 4th parameter인 less function의 정렬기준에 따라 3rd parameter로 끝나는 1st parameter, 2nd parameter, 3rd parameter의 병합구조를 생성해준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_sort (struct list \*,  list\_less\_func \*, void \*aux) |
| **Parameter** | 1. sort를 할 list 2. list\_elem간의 대소비교를 위한 function 3. 보조 parameter |
| **Return** | void |
| **Function** | 1st parameter인 list내부의 list\_elem들을 2nd parameter(user defined function)를 사용하여 정렬기준을 정해주고(대소비교), 정렬을 진행해준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert\_ordered (struct list \*, struct list\_elem \*,  list\_less\_func \*, void \*aux) |
| **Parameter** | 1. insert를 진행할 list 2. insert할 list\_elem 3. list\_elem간의 대소비교를 위한 function 4. 보조 parameter |
| **Return** | void |
| **Function** | 1st parameter인 list내부에 2nd parameter인 list\_elem을 적절한 위치에 삽입한다. list\_elem간의 대소비교를 위한 3rd parameter(user defined function)와 iteration을 이용해 적절한 자리를 찾고, insertion이 된다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_unique (struct list \*, struct list \*duplicates,  list\_less\_func \*, void \*aux) |
| **Parameter** | 1. unique한 list\_elem들만을 남길 list 2. 제거된 list\_elem들을 저장하는 list 3. list\_elem간의 대소비교(=)를 위한 function 4. 보조 parameter |
| **Return** | void |
| **Function** | 1st parameter인 list내부에 존재하는 list\_elem들중 해당하는 list\_item의 data가 중복된 list\_elem들을 전부 제거해준뒤, 2nd parameter인 duplicates list에 insertion해준다. 만약 ./testlib에서 stdin으로 한 개의 list만이 입력될 경우 duplicates list를 전부 free해주고, 두 개의 list가 들어올 경우 duplicates list를 두 번째input list로 설정해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_max (struct list \*, list\_less\_func \*, void \*aux) |
| **Parameter** | 1. max값을 찾고싶은 list 2. list\_elem간의 대소비교를 위한 function  3. 보조 parameter |
| **Return** | struct list\_elem \* (해당하는 list\_item의 data가 가장 작은 list\_elem) |
| **Function** | 1st parameter인 list의 내부에서 iteration과 대소비교 function(user defined function)을 통해 각각의 list\_elem들중 해당하는 list\_item의 data가 가장 큰 list\_elem을 찾은 뒤, 이를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem \*list\_min (struct list \*, list\_less\_func \*, void \*aux) |
| **Parameter** | 1. min값을 찾고싶은 list 2. list\_elem간의 대소비교를 위한 function  3. 보조 parameter |
| **Return** | struct list\_elem \* (해당하는 list\_item의 data가 가장 작은 list\_elem) |
| **Function** | 1st parameter인 list의 내부에서 iteration과 대소비교 function(user defined function)을 통해 각각의 list\_elem들중 해당하는 list\_item의 data가 가장 작은 list\_elem을 찾은 뒤, 이를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem \*, struct list\_elem \*) |
| **Parameter** | 1. 위치를 바꿔줄 첫번째 list\_elem 2. 위치를 바꿔줄 두번째 list\_elem |
| **Return** | void |
| **Function** | list내부의 두개의 list\_elem을 swap해주는 기능을 한다. 두 리스트가 인접해 있을 경우, 그저 elem을 swap해주는 것이 아닌 prev와 next도 고려해서 swap을 해준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*) |
| **Parameter** | 1. shuffle을 진행할 list |
| **Return** | void |
| **Function** | list내부의 list\_elem들을 rand함수와 구현해둔 list\_swap function을 이용하여 무작위로 정렬해준다. |

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | #define hash\_entry(HASH\_ELEM, STRUCT, MEMBER) \  ((STRUCT \*) ((uint8\_t \*) &(HASH\_ELEM)->list\_elem \  - offsetof (STRUCT, MEMBER.list\_elem))) |
| **Parameter** | 1. hash\_item을 얻고싶은 hash\_elem 2. struct hash\_item(내장되어있는 구조) 3. elem(member) |
| **Return** | 입력받은 hash\_elem이 내장되어있는 hash\_item struct(2nd parameter) |
| **Function** | 1st parameter로 hash\_elem을 입력받고 outer structure인 2nd parameter와 member인 3rd parameter를 통해 그 hash\_elem이 내장되어 있는 2nd parameter형태를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | #define list\_elem\_to\_hash\_elem(LIST\_ELEM) \  list\_entry(LIST\_ELEM, struct hash\_elem, list\_elem) |
| **Parameter** | 1. hash\_elem을 얻고싶은 list\_elem 2. struct hash\_elem(내장되어있는 구조) 3. elem(member) |
| **Return** | 입력받은 list\_elem이 내장되어있는 hash\_elem struct(2nd parameter) |
| **Function** | 1st parameter로 list\_elem을 입력받고 outer structure인 2nd parameter와 member인 3rd parameter, list\_entry macro를 이용하여 1st parameter인 list\_elem이 내장된 hash\_elem struct를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_init (struct hash \*, hash\_hash\_func \*, hash\_less\_func \*, void \*aux) |
| **Parameter** | 1. initial작업을 해줄 hash 2. hash함수를 이용해서 계산한 hash값(unsigned) 3. hash\_elem간의 대소비교를 위한 function 4. 보조 parameter |
| **Return** | bool : true(initial이 성공했을 때), false(initial이 실패했을 때) |
| **Function** | hash table을 초기화한뒤, hash구조체 내부의 hash function으로 2nd parameter를 설정하고, hash\_elem들간의 대소비교를 위한 function으로 3rd parameter를 넣어준다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear (struct hash \*, hash\_action\_func \*) |
| **Parameter** | 1. 모든 hash\_elem을 제거할 hash table 2. 특정 hash\_elem을 제거할 함수(void) |
| **Return** | void |
| **Function** | user defined function인 2nd Parameter와 iteration을 통해 1st parameter로 입력받은 hash table내부의 hash\_elem들을 전부 제거해준다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy (struct hash \*, hash\_action\_func \*) |
| **Parameter** | 1. 제거할 hash table 2. 특정 hash\_elem을 제거할 함수(void) |
| **Return** | void |
| **Function** | user defined function인 2nd parameter와 iteration을 통해 1st parameter로 입력받은 hash\_table내부의 hash\_elem들을 전부 제거해준뒤 buckets도 함께 제거해준다. hash\_clear function과의 차이점은 hash\_clear는 hash table내부의 hash\_elem들만 제거하고 buckets은 남겨놓는 반면 hash\_delete는 buckets도 제거해줌으로서 hash table 자체를 제거해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_insert (struct hash \*, struct hash\_elem \*) |
| **Parameter** | 1. 입력받은 hash\_elem을 insert해줄 hash table 2. insert해줄 hash\_elem |
| **Return** | struct hash\_elem \*(duplicate한 경우 inseert하지 않고 반환) or null pointer(duplicate한 data가 없을 경우) |
| **Function** | 입력받은 1st parameter인 hash table 내부에 2nd parameter인 hash\_elem을 insert해준다. 만약 이미 hash table에 hash\_item의 data가 duplicate한 hash\_elem이 존재할경우 insert하지 않고 반환해주고, 아닌 경우 null pointer를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_replace (struct hash \*, struct hash\_elem \*) |
| **Parameter** | 1. 입력받은 hash\_elem을 대체할 hash table 2. replace해줄 hash\_elem |
| **Return** | struct hash\_elem \* (만약 hash\_elem이 대체될 경우 대채된 hash\_elem을 반환한다) |
| **Function** | 입력받은 1st parameter인 hash table 내부에 2nd parameter인 hash\_elem을 insert해준뒤 hash\_item의 data값이 duplicate한 hash\_elem이 존재한다면 그것을 제거해주고 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_find (struct hash \*, struct hash\_elem \*) |
| **Parameter** | 1. 입력받은 hash\_elem을 찾아줄 hash table 2. 찾는 hash\_elem |
| **Return** | struct hash\_elem \* (입력받은 hash\_elem을 찾은 경우) or null pointer(입력받은 hash\_elem을 찾지 못한 경우) |
| **Function** | 입력받은 1st parameter인 hash table 내부에 2nd parameter인 hash\_elem이 존재한다면 해당하는 hash\_elem을 반환해주고 찾지 못한 경우 null pointer를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_delete (struct hash \*, struct hash\_elem \*) |
| **Parameter** | 1. 입력받은 hash\_elem을 제거할 hash table 2. 제거할 hash\_elem |
| **Return** | struct hash\_elem \* (입력받은 hash\_elem을 찾은경우) or null pointer(입력받은 hash\_elem을 찾지 못한 경우) |
| **Function** | 입력받은 1st parameter인 hash table 내부에서 2nd parameter인 hash\_elem을 찾은 경우 제거해주고 이를 반환한다. 만약 찾지 못한 경우 null pointer를 반환하게 된다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_apply (struct hash \*, hash\_action\_func \*) |
| **Parameter** | 1. 입력받은 action취해줄 hash table 2. action function(void) |
| **Return** | void |
| **Function** | 입력받은 1st parameter인 hash table 내부의 hash\_elem들을 hash\_entry를 통해 data를 받아온 뒤 2nd parameter인 action(square,triple) function을 통해 data들에 action을 취해준다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_first (struct hash\_iterator \*, struct hash \*) |
| **Parameter** | 1. hash table iterator 2. first를 참조하고 싶은 hash table |
| **Return** | void |
| **Function** | hash iterator를 설정해주는 함수로서 hash table의 element들을 iterator struct에 initial해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_next (struct hash\_iterator \*) |
| **Parameter** | 1. hash table iterator |
| **Return** | struct hash\_elem \* (현재 hash\_iterator의 다음 hash\_elem) or null pointer(다음 hash\_elem이 존재하지 않을 경우) |
| **Function** | 1st parameter인 hash\_iterator를 다음 hash\_elem으로 이동한뒤 해당하는 hash\_elem을 반환한다. 만약 다음 hash\_elem가 존재하지 않을 경우 null pointer를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_cur (struct hash\_iterator \*) |
| **Parameter** | 1. hash table iterator |
| **Return** | struct hash\_elem \* (현재 hash table iterator가 방문하고 있는 hash\_elem) or null pointer(end of hash table일 경우) |
| **Function** | 1st parameter인 hash table iterator가 현재 방문하고 있는 hash\_elem을 반환한다. 만약 현재 방문하고 있는 hash\_elem이 hash table의 마지막 위치일 경우 null pointer를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t hash\_size (struct hash \*) |
| **Parameter** | 1. size를 구하고 싶은 hash table |
| **Return** | size\_t(unsigned int) |
| **Function** | hash struct내부의 elem\_cnt를 통해 hash table내부의 hash\_elem개수를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | bool hash\_empty (struct hash \*) |
| **Parameter** | 1. 비어있는지를 확인하고 싶은 hash table |
| **Return** | bool : true(hash\_elem이 하나도 없을 경우), false(hash\_elem이 하나라도 있을 경우) |
| **Function** | hash struct내부의 elem\_cnt를 통해서 elem\_cnt가 0일 경우 hash\_elem이 하나도 없으므로 true를 반환, 0이 아닐 경우 hash\_elem이 한 개이상 있으므로 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_bytes (const void \*, size\_t) |
| **Parameter** | 1. 2nd parameter크기의 bytes 2. 1st parameter의 size |
| **Return** | unsigned (hash value) |
| **Function** | 2nd parameter크기의 byte값을 1st parameter로 받아온뒤, 각 byte마다 연산을 취해줘 hash value를 만든다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_string (const char \*) |
| **Parameter** | 1. char pointer type |
| **Return** | unsigned (hash value) |
| **Function** | char 배열 형식의 문자열을 1st parameter로 받아 그에 해당하는 hash value를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int (int) |
| **Parameter** | 1. integer hash function을 적용할 integer value |
| **Return** | unsigned (hash value) |
| **Function** | 앞서 말한 hash\_bytes function을 이용하여 1st parameter인 integer값의 hash value를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | static struct list \*  find\_bucket (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | 1. bucket이 속한 hash table 2. bucket에 속한 hash\_elem |
| **Return** | static struct list \* (bucket) |
| **Function** | 2nd parameter를 통해 bucket의 index를 알아낸 이후에 hash table의 index번째 bucket을 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | static struct hash\_elem \*  find\_elem (struct hash \*h, struct list \*bucket, struct hash\_elem \*e) |
| **Parameter** | 1. bucket이 속한 hash table 2. hash\_elem이 속해있는지 검색할 bucket  3. 찾을 hash\_elem |
| **Return** | struct hash\_elem \*  (hash\_elem이 속해있을 경우 hash\_elem반환) or null pointer(hash\_elem을 못찾은경우) |
| **Function** | 입력받은 1st parameter hash table내부의 2nd paramter bucket안에 3rd parameter인 hash\_elem이 존재하는지 확인하고 존재할 경우 이를 반환, 아닐 경우 null pointer를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t  turn\_off\_least\_1bit (size\_t x) |
| **Parameter** | 1. 최하위 비트를 off해주고 싶은 value |
| **Return** | static inline size\_t (최하위 비트가 ) |
| **Function** | 입력받은 value의 1을 뺀값과 입력받은 값을 &연산 해줌으로서 최하위 비트를 off해준채로 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t  is\_power\_of\_2 (size\_t x) |
| **Parameter** | 1. 2의 n승인지 확인하고 싶은 value |
| **Return** | static inline size\_t : true(2의 n승인 경우), false(2의 n승이 아닌 경우) |
| **Function** | 만약 입력받은 1st parameter가 2의 n승인 경우 true를 반환, 아닌 경우 false를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | static void  rehash (struct hash \*h) |
| **Parameter** | 1. bucket의 size를 바꾸고 싶은 hash table |
| **Return** | static void |
| **Function** | 입력받은 hash table의 bucket size를 현재의 hash table의 이상적인 조건에 맞춰서 바꿔준다. |

|  |  |
| --- | --- |
| **Prototype** | static void  insert\_elem (struct hash \*h, struct list \*bucket, struct hash\_elem \*e) |
| **Parameter** | 1. bucket이 존재하는 hash table 2. hash\_elem을 insert할 bucket  3. insert할 hash\_elem |
| **Return** | static void |
| **Function** | 3rd parameter인 hash\_elem을 1st parameter인 hash table안에 존재하는 2nd parameter bucket에 insert해준다. |

|  |  |
| --- | --- |
| **Prototype** | static void  remove\_elem (struct hash \*h, struct hash\_elem \*e) |
| **Parameter** | 1. bucket이 존재하는 hash table 2. remove할 hash\_elem |
| **Return** | static void |
| **Function** | 2nd parameter인 hash\_elem을 1st parameter hash table에서 제거해준다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int) |
| **Parameter** | 1. user defined hash function을 적용할 integer value |
| **Return** | unsigned (hash value) |
| **Function** | interger값의 대표적인 해시함수인 Knuth's multiplicative 방법을 이용하여 해시값을 생성하고 return해준다. |

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t  elem\_idx (size\_t bit\_idx) |
| **Parameter** | 1. index를 알고 싶은 bit\_idx |
| **Return** | static inline size\_t (index) |
| **Function** | 1st parameter가 포함된 element의 index를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | static inline elem\_type  bit\_mask (size\_t bit\_idx) |
| **Parameter** | 1. elem\_type을 구할 bit\_idx |
| **Return** | static inline elem\_type (bitmap->elem\_type) |
| **Function** | 입력받은 1st parameter에 적합한 struct bitmap내부의 element인 elem\_type을 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t  elem\_cnt (size\_t bit\_cnt) |
| **Parameter** | 1. 필요한 element의 개수를 구할 bit\_cnt |
| **Return** | static inline size\_t (element개수) |
| **Function** | DIV\_ROUND\_UP function과 ELEM\_BITS를 이용하여 입력받은 bit\_cnt에 필요한 element 개수를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | static inline size\_t  byte\_cnt (size\_t bit\_cnt) |
| **Parameter** | 1. 필요한 Byte수를 구할 bit\_cnt |
| **Return** | static inline size\_t (Byte수) |
| **Function** | struct bitmap내부의 elem\_type과 1st parameter를 이용하여 입력받은 bit\_cnt에 필요한 Byte수를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | static inline elem\_type  last\_mask (const struct bitmap \*b) |
| **Parameter** | 1. bitmask를 얻고 싶은 bitmap |
| **Return** | static inline elem\_type (bitmask) |
| **Function** | 입력받은 1st parameter의 비트중 마지막 element에 사용된 비트만 1로 set되고 나머지는 0인 bitmask를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create (size\_t bit\_cnt) |
| **Parameter** | 1. initial할 bitmap의 bit size |
| **Return** | struct bitmap \* (bit\_cnt 크기의 bit수를 가지는 bitmap) |
| **Function** | unsigned int크기의 bit\_cnt를 1st parameter로 입력받아 그 크기만큼의 bit수를 가지는 bit map을 만들고, 모두 false로 초기화를 해준뒤 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create\_in\_buf (size\_t bit\_cnt, void \*, size\_t byte\_cnt) |
| **Parameter** | 1. initial할 bitmap의 bit size 2. preallocated된 메모리공간 3. 2nd parameter의 size |
| **Return** | struct bitmap \* (bit\_cnt 크기의 bit수를 가지는 bitmap) |
| **Function** | 3rd parameter크기의 미리 할당된 byte공간(2nd parameter)에서 1st parameter 크기의 bit map을 생성하여 반환해준다. 그러므로, 3rd parameter의 값이 무조건 1st parameter의 값보다 커야한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_buf\_size (size\_t bit\_cnt) |
| **Parameter** | 1. bitmap의 bit수 |
| **Return** | size\_t(unsigned int) (변환된 byte수) |
| **Function** | 앞서 말한 bitmap\_create\_in\_buf함수에서 사용되는 함수로서 bit map의 크기인 1st parameter크기의 bit를 byte수로 변환해주고 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy (struct bitmap \*) |
| **Parameter** | 1. 제거할 bitmap |
| **Return** | void |
| **Function** | 1st parameter로 입력받은 bitmap을 free함수를 통해 동적할당 해제를 해준다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_size (const struct bitmap \*) |
| **Parameter** | 1. size를 구하고 싶은 bitmap |
| **Return** | size\_t(unsigned int) (bitmap의 bit수) |
| **Function** | bitmap struct 내부에 존재하는 bit\_cnt element를 통해 1st parameter로 입력받은 bitmap의 bit수를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set (struct bitmap \*, size\_t idx, bool) |
| **Parameter** | 1. 하나의 bit를 바꿔줄 bitmap 2. 바꿔줄 bit의 bitmap에서의 index  3. 바꿔줄 value(bool) |
| **Return** | void |
| **Function** | 입력받은 1st parameter인 bitmap내부에서 2nd parameter의 index를 가지는 bit를 3rd parameter의 값으로 바꿔준다. 만약 3rd parameter가 true일 경우 bitmap\_mark function을 통해서 initial을 해주고 false일 경우 bitmap\_reset function을 통해서 initial을 해준다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark (struct bitmap \*, size\_t idx) |
| **Parameter** | 1. 하나의 bit를 true로 바꿔줄 bitmap 2. 바꿔줄 bit의 bitmap에서의 index |
| **Return** | void |
| **Function** | 입력받은 1st parameter인 bitmap내부에서 2nd parameter의 index를 가지는 bit를 true, 즉 1로 바꿔준다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset (struct bitmap \*, size\_t idx) |
| **Parameter** | 1. 하나의 bit를 false로 바꿔줄 bitmap 2. 바꿔줄 bit의 bitmap에서의 index |
| **Return** | void |
| **Function** | 입력받은 1st parameter인 bitmap내부에서 2nd parameter의 index를 가지는 bit를 false, 즉 0으로 바꿔준다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip (struct bitmap \*, size\_t idx) |
| **Parameter** | 1. 하나의 bit를 toggle해줄 bitmap 2. toggle해줄 bit의 bitmap에서의 index |
| **Return** | void |
| **Function** | 입력받은 1st parameter인 bitmap내부에서 2nd parameter의 index를 가지는 bit를 toggle해준다. 즉, true일 경우 false로 만들고, false일 경우 true로 만들어준다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_test (const struct bitmap \*, size\_t idx) |
| **Parameter** | 1. 하나의 bit값을 받아올 bitmap 2. 값을 받아올 bit의 bitmap에서의 index |
| **Return** | bool : true(bit값이 1인 경우), false(bit값이 0인 경우) |
| **Function** | 입력받은 1st parameter인 bitmap내부에서 2nd parameter의 index를 가지는 bit값을 return해준다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all (struct bitmap \*, bool) |
| **Parameter** | 1. all\_set할 bitmap 2. set할 value(bool) |
| **Return** | void |
| **Function** | 입력받은 1st parameter인 bitmap의 모든 bit값을 bitmap\_set\_multiple function을 이용하여 2nd parameter value(true or false)로 변경해준다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple (struct bitmap \*, size\_t start, size\_t cnt, bool) |
| **Parameter** | 1. bit값을 변경해줄 bitmap 2. 변경 시작 bit의 bitmap에서의 index  3. 변경할 bit개수 4. set할 value(bool) |
| **Return** | void |
| **Function** | 입력받은 1st parameter인 bitmap의 2nd parameter index의 bit를 시작점으로 3rd parameter개수만큼의 bit를 4th parameter value로 set해준다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (const struct bitmap \*, size\_t start, size\_t cnt, bool) |
| **Parameter** | 1. bit의 개수를 구할 bitmap 2. 검색시작 bit의 bitmap에서의 index  3. 검색할 bit개수 4. 검색할 value |
| **Return** | size\_t (value값을 만족하는 [start, start+cnt) 범위의 bit개수) |
| **Function** | 입력받은 1st parameter인 bitmap의 2nd parameter index의 bit를 시작점으로 3rd parameter개수만큼의 bit들중 4th parameter value와 같은 값을 가지는 bit수를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_contains (const struct bitmap \*, size\_t start, size\_t cnt, bool) |
| **Parameter** | 1.만족하는bit가 있는지 구할 bitmap 2. 검색시작 bit의 bitmap에서의 index  3.검색할 bit개수 4. 검색할 value |
| **Return** | bool : true(범위내 value값을 가지는 bit가 하나라도 있을 경우), false(범위내value값을 가지는 bit가 하나도 없을 경우) |
| **Function** | 입력받은 1st parameter인 bitmap의 2nd parameter index의 bit를 시작점으로 3rd parameter개수만큼의 bit들중 4th parameter value와 같은 값을 가지는 bit가 하나라도 있을경우 true(1)을 반환하고, 아닐 경우 false(0)을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_any (const struct bitmap \*, size\_t start, size\_t cnt) |
| **Parameter** | 1. true인 bit가 있는지 구할 bitmap 2. 검색시작 bit의 bitmap에서의 index  3. 검색할 bit개수 |
| **Return** | bool : true(범위내 true값을 가지는 bit가 하나라도 있을 경우), false(범위내 true값을 가지는 bit가 하나도 없을 경우) |
| **Function** | bitmap\_contains function을 이용하여 입력받은 1st parameter인 bitmap의 2nd parameter index의 bit를 시작점으로 3rd parameter개수만큼의 bit들중 true값을 가지는 bit가 하나라도 있으면 true를, 없으면 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_none (const struct bitmap \*, size\_t start, size\_t cnt) |
| **Parameter** | 1. true인 bit가 있는지 구할 bitmap 2. 검색시작 bit의 bitmap에서의 index  3. 검색할 bit개수 |
| **Return** | bool : true(범위내 true을 가지는 bit가 하나도 없을 경우), false(범위내 true값을 가지는 bit가 하나라도 있을 경우) |
| **Function** | bitmap\_contains function을 이용하여 입력받은 1st parameter인 bitmap의 2nd parameter index의 bit를 시작점으로 3rd parameter개수만큼의 bit들중 true값을 가지는 bit가 하나라도 있으면 false를, 없으면 true를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_all (const struct bitmap \*, size\_t start, size\_t cnt) |
| **Parameter** | 1.모든bit가 true인지 검색할 bitmap 2. 검색시작 bit의 bitmap에서의 index 3.검색할 bit개수 |
| **Return** | bool : true(범위내 모든 bit가 true인 경우), false(범위내 false값을 가지는 bit가 하나라도 있을 경우) |
| **Function** | bitmap\_contains function을 이용하여 입력받은 1st parameter인 bitmap의 2nd parameter index의 bit를 시작점으로 3rd parameter개수만큼의 bit들중 모든 bit가 true값을 가지면 true를 반환하고, 아닐 경우 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan (const struct bitmap \*, size\_t start, size\_t cnt, bool) |
| **Parameter** | 1. 값을 만족하는 연속한 bit의 위치를 구할 bitmap 2. 검색시작 bit의 bitmap에서의 index 3. 연속하는 개수 4. 검색할 value(bool) |
| **Return** | size\_t (연속하는 bit의 시작지점 index) or BITMAP\_ERROR |
| **Function** | 입력받은 1st parameter인 bitmap의 2nd parameter index의 bit를 시작점으로 3rd parameter개수만큼 4th parameter의 value를 연속으로 가지고 있는 비트열이 존재할 경우, 그 첫번째 index를 반환한다. 만약 만족하는 비트열이 존재하지 않을경우 BITMAP\_ERROR를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_scan\_and\_flip (struct bitmap \*, size\_t start, size\_t cnt, bool) |
| **Parameter** | 1. 값을 만족하는 연속한 bit의 위치를 구할 bitmap 2. 검색시작 bit의 bitmap에서의 index 3. 연속하는 개수 4. 검색할 value(bool) |
| **Return** | size\_t (연속하는 bit의 시작지점 index) or BITMAP\_ERROR |
| **Function** | 앞서 말한 bitmap\_scan function에 1st, 2nd, 3rd, 4th parameter를 넘겨주어 결과값을 저장한뒤에 그 결과값이 BITMAP\_ERROR가 아닌경우(즉, 만족하는 비트열이 존재할 경우) bimap\_set\_multiple function을 이용하여 만족하는 비트열을 모두 반전해준뒤에 그 첫 번째 index를 반환한다. 만약 만족하는 비트열이 존재하지 않을경우 BITMAP\_ERROR를 그대로 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_file\_size (const struct bitmap \*b) |
| **Parameter** | 1. file에 저장하고 싶은 bitmap |
| **Return** | size\_t (bitmap의 저장에 필요한 Byte수) |
| **Function** | bitmap을 입력으로 받아 해당하는 bitmap을 파일에 저장하기 위한 Byte수를 반환해준다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_read (struct bitmap \*b, struct file \*file) |
| **Parameter** | 1. file에서 read할 bitmap 2. read할 file |
| **Return** | bool : true(정상적으로 file을 읽어올 경우), false(읽어오지 못한 경우) |
| **Function** | 2nd parameter file을 통해 bitmap을 읽어와 1st parameter에 저장한다. 성공적으로 read한 경우 true를 반환하고, 아닌 경우 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_write (const struct bitmap \*b, struct file \*file) |
| **Parameter** | 1. file에 write할 bitmap 2. write할 file |
| **Return** | bool : true(정상적으로 file에 write가 된 경우), false(정상적인 write를 하지 못한 경우) |
| **Function** | 2nd parameter file에 1st parameter bitmap을 write한다. 성공적으로 file에 write가 된 경우 true를 반환하고, 아닌 경우 false를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_dump (const struct bitmap \*) |
| **Parameter** | 1. 16진수형태로 출력할 bitmap |
| **Return** | void |
| **Function** | 2진수 형태로 저장되어 있는 bitmap을 16진수 형태로 출력해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_expand(struct bitmap \*, int) |
| **Parameter** | 1. 확장할 bitmap 2. 확장하고 싶은 추가 size |
| **Return** | struct bitmap \* (2nd parameter만큼 추가로 확장된 bitmap) |
| **Function** | 기존의 1st parameter bitmap size에 2nd parameter를 더한만큼 새로운 bitmap을 만들어주고 1st parameter bitmap size만큼 복사해준다. 이후 기존 bitmap을 bitmap\_destroy function을 이용하여 free해준뒤에 확장된 bitmap을 반환해준다. |